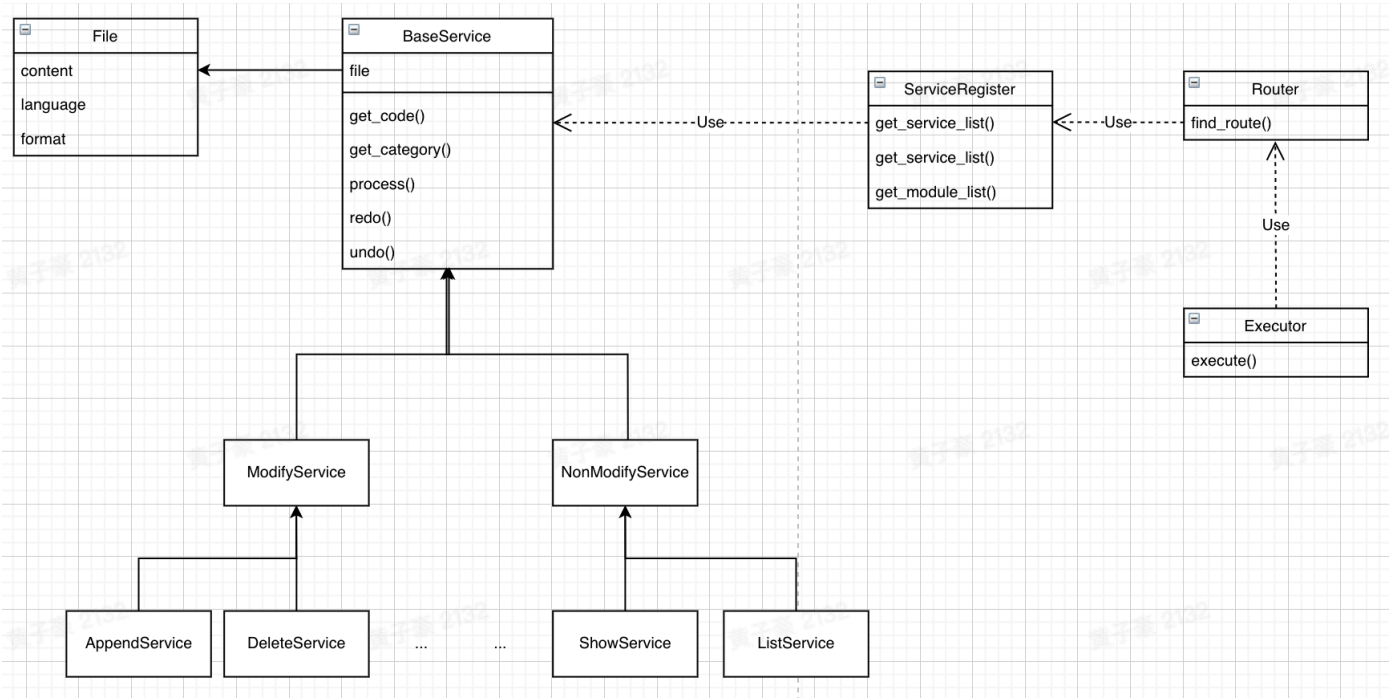
简易文本编辑器

By Huang Zihao 18302010034

程序结构

程序图：https://bytedance.feishu.cn/docx/doxcnggJcKeAh5ZtxCJZybTSVdh



Everything is object. 在我的设计中，不仅具体的file是对象，连路由模块Router、ServiceRegister和每条指令都是一个对象。

每条指令都继承自抽象的BaseService，并implements它的五个函数。

- get_code用于路由模块
- get_category用于判断需不需要redo和undo
- process每条命令处理具体的工作
- undo和redo负责对于修改类命令undo和redo指令对象自己

ServiceRegister通过反射技术获取BaseService的所有子类，并提供给路由模块调用。

Route根据指令的开头判断调用BaseService的哪个实例。

设计模式

单例模式

修改的文件对象采用了单例模式，确保了所有service对象修改的是同一个文件。

装饰器模式

实现了装饰器singleton_wrapper和exception_wrapper，因为很多场景下都需要单例模式和异常处理，使用装饰器模式能极少地减少对程序代码的侵入性，更加模块化。

观察者模式

宏命令的实现刚开始着实令我费了一些脑筋，因为ServiceRegister通常都是程序初始化的时候主动的去获取所有的服务指令的，如何在程序运行中动态地添加新的指令呢？我的解决方案是：不是ServiceRegister定时地去“拉取”所有服务指令，而是把ServiceRegister设置成观察者，每当服务指令更新时去通知ServiceRegister。

我的设计模式的优点

1. 便于扩展，这是最大的优点。当新增指令时，只需要在service模块下新增一个service子类，实现它的5个接口（每一个的实现都很简单，只需要关注指令本身实现功能！）
2. 便于理解，任何看过代码的人能很快捷地理解这个项目

Why Python Not Java?

python和java都是面向对象的语言，通过我的程序结构图可以很清楚的发现在我的程序中oop的思想无处不在：

- 万物皆对象。每种指令、Router、ServiceRegister和File都是对象
- 继承。所有指令继承自BaseService

而选择python的原因是因为python中使用装饰器和反射非常方便：可以通过注解的形式方便地引用一个装饰器，可以通过module模块相关api方便地实现ServiceRegister。

所以我选择python，这可以很好地实现我的程序结构，体现oop的思想，并且python有着方便的装饰器和反射。

For TA: How To Test?

运行项目根目录下的test.py文件即可测试**自动测试用例**和**5个手工测试用例**，如果要测试新的测试用例，只需要替换掉测试的文本和字典即可

自动测试用例说明

AutoTestCase.txt

```
a "What is your point "
A "of view"
s
D 5
u
s
r
s
u
s
d 5
u
s
l 10
s
m 2 m2
s
$m2
lang eng
content txt
spell
spell-a
spell-m

```

这个测试用例覆盖了所有的指令，包括A a s D d u r l m lang content spell spell-a spell-m